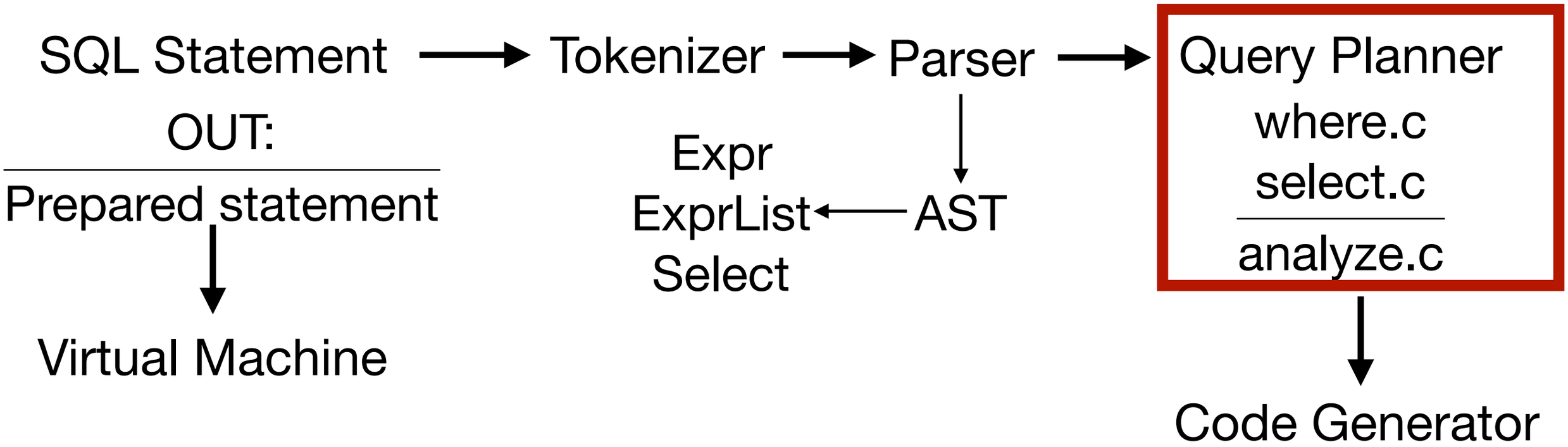
HighLoad++ 2017

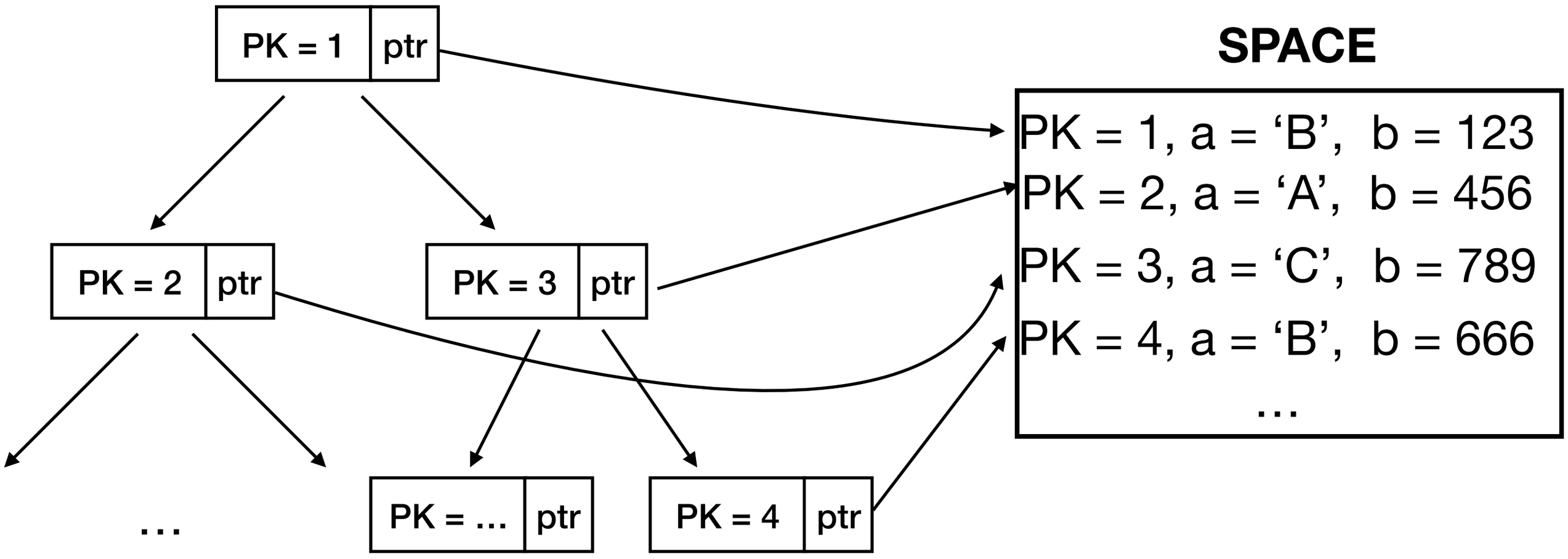
Оптимизация запросов SQL в Tarantool

Никита Петтик

Ход выполнения запроса

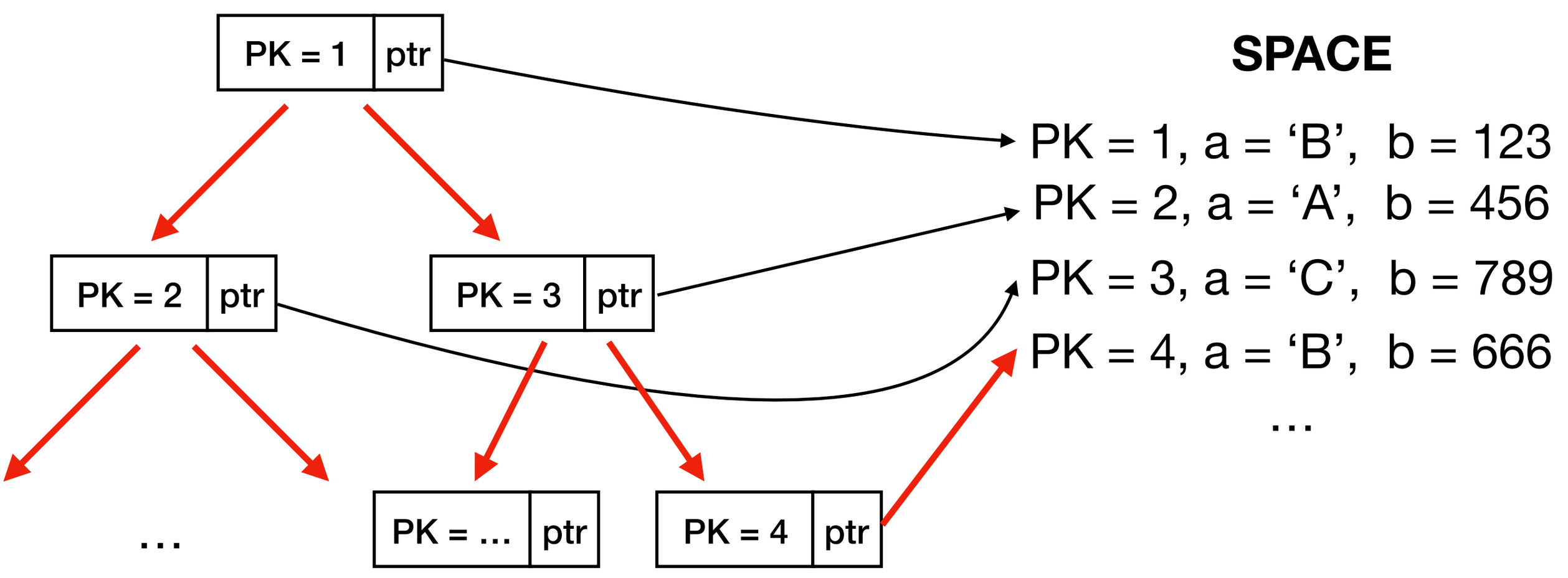


Таблицы

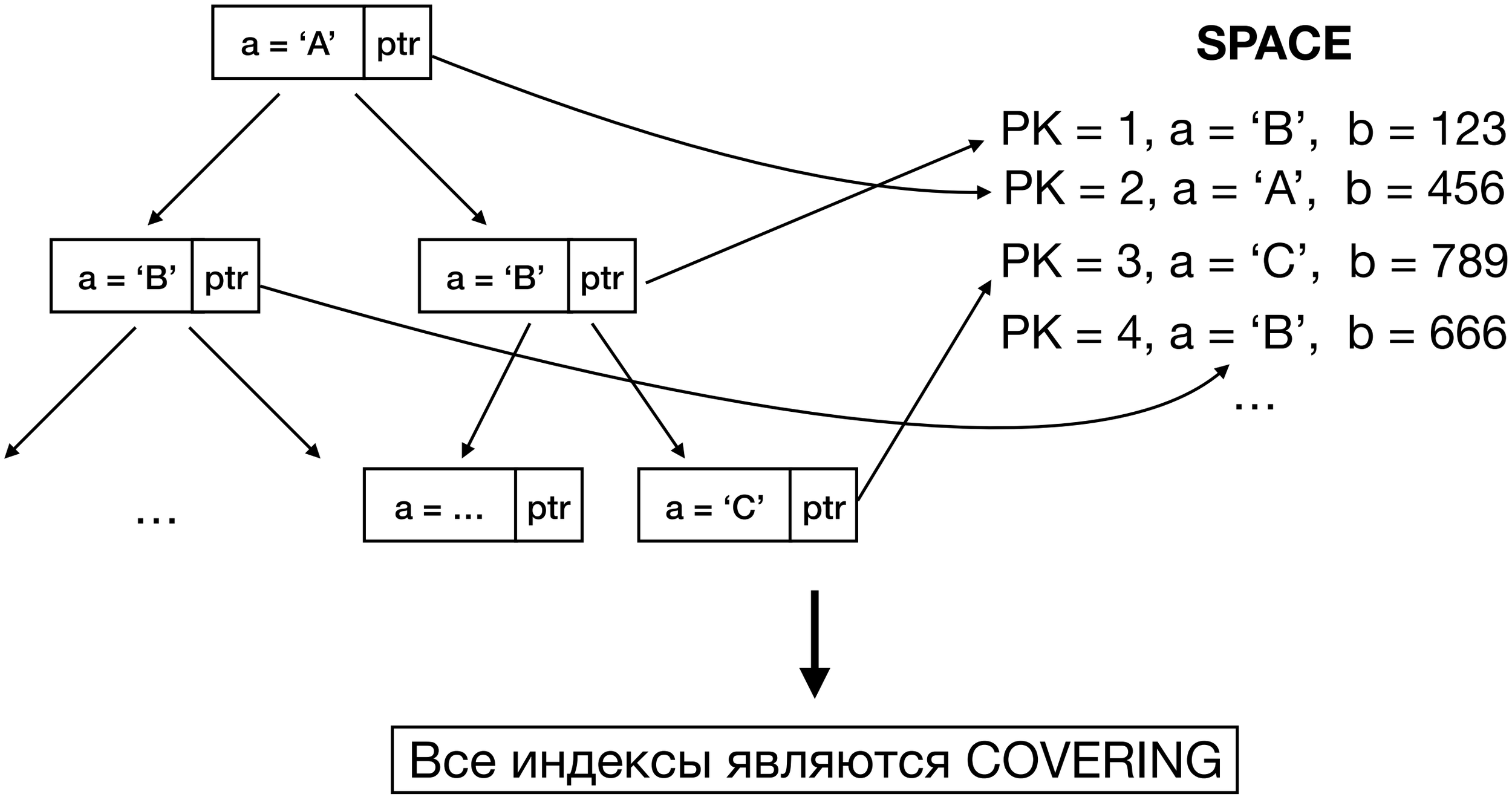

PK = 1
ptr
PK = 2
ptr
PK = 3
ptr
...
PK = ...
ptr
PK = 4
ptr
SPACE
PK = 1, a = ‘B’, b = 123
PK = 2, a = ‘A’, b = 456
PK = 3, a = ‘C’, b = 789
PK = 4, a = ‘B’, b = 666
...

Поиск по таблице TABLE SCAN

```
SELECT a WHERE b = 666;
SELECT * WHERE PK = 4;
```



Индексы



Все индексы являются COVERING

AND

SELECT price FROM Fruits WHERE fruit = ‘Orange’ AND city = ‘NY’

fruit	id
Apple	2
Banana	6
Grape	4
Lemon	5
Orange	1
Orange	7
Peach	3

city	id
Kiev	1
Kiev	5
MILAN	3
MSK	2
MSK	6
NY	4
NY	7

id	fruit	city	price
1	Orange	KIEV	85
2	Apple	MSK	45
3	Peach	MILAN	60
4	Grape	NY	80
5	Lemon	KIEV	125
6	Banana	MSK	245
7	Orange	NY	105

OR

SELECT price FROM Fruits WHERE fruit = ‘Orange’ OR city = ‘NY’

fruit	id
Apple	2
Banana	6
Grape	4
Lemon	5
Orange	1
Orange	7
Peach	3

city	id
Kiev	1
Kiev	5
MILAN	3
MSK	2
MSK	6
NY	4
NY	7

UNION

id	fruit	city	price
1	Orange	KIEV	85
2	Apple	MSK	45
3	Peach	MILAN	60
4	Grape	NY	80
5	Lemon	KIEV	125
6	Banana	MSK	245
7	Orange	NY	105

Поиск оптимального плана

SELECT * FROM t1 INNER JOIN t2 WHERE

t1.x > 5 AND t1.y = 3 AND t2.z = 5;

Term Term Term

Loop Loop

Можно рассматривать Loop’ы как вершины графа, где ребра имеют стоимость.
Задача — найти наименьший по стоимости путь длинной равной кол-ву условий в запросе.

Сбор статистики

- ANALYZE;
- _sql_stat1(tbl, idx, stat)
- _sql_stat4(tbl, idx, nEq, nLt, nDLt, sample)
- Можно управлять вручную: noskipscan, unordered

_sql_stat1

_sql_stat1(tbl, idx, stat)

tbl - таблица
idx - индекс
stat - статистика

CREATE INDEX i ON t(a,b,c)
stat = “8 3 2 1”

Строк в индексе

Среднее количество совпадающих срок в первом столбце

Во втором и третьем (вместе)

id	a	b	c
1	1	1	1
2	1	1	2
3	1	2	2
4	1	2	3
5	2	2	2
6	2	2	3
7	2	3	3
8	3	3	3

_sql_stat4

- _sql_stat4(tbl, idx, nEq, nLt, nDLt, sample)
- sample — значение ключа индекса
- Для каждого индекса не более 24 сэмплов
- Выбираются лучшие сэмлы: сэмпл лучше, если содержит больше nEq элементов
- nEq — кол-во строк в левом столбце индекса, значение которых совпадает с первым значением ключа sample’a

_sql_stat4

CREATE INDEX i ON t(a,b,c);

sample = ‘1 2 2’

nEq = ‘4 2 2’
nLt = ‘0 2 2’
nDLt = ‘0 1 2’

sample = ‘2 3 3’

nEq = ‘3 1 1’
nLt = ‘4 6 6’
nDLt = ‘1 3 6’

id	a	b	c
1	1	1	1
2	1	1	2
3	1	2	2
4	1	2	2
5	2	2	2
6	2	2	3
7	2	3	3
8	3	3	3

Оптимизации без статистики

- WHERE x < ? уменьшает кол-во строк для поиска в 4 раза

- WHERE x>? AND x<? в 64 раза

- Таблица в среднем содержит около 260000 строк

Выбор лучшего индекса

- Для каждого индекса рассчитывается стоимость выполнения в логарифмической шкале (10log2)

- Стоимость состоит из трех оценок:

1. rSetup — стоимость запуска: сортировка, авто-индексы
2. rRun — стоимость одного цикла
   rRun = 10*log(cost))
   cost = nRow * 3 — TABLE SCAN
   cost = nRow * 4 — SCAN OF COVERING INDEX
3. nOut — количество просмотренных строк

Выбор лучшего индекса

CREATE TABLE t1(id PRIMARY KEY, x, y);

x, y IN [0; 100] — равномерно распределены

CREATE INDEX t1x ON t1(x);
CREATE INDEX t1y ON t1(y);

SELECT * FROM t1 WHERE x > 0 AND x < 100 AND y > 99;

Выбор лучшего индекса

До ANALYZE:

```
t1.t1y                          0 f 00262 N 1 cost 0,188,178
```

```
t1.t1x                          0 f 00272 N 2 cost 0,148,139
```

После ANALYZE:

```
STAT4 range scan: 97..100  est=16
Range scan lowers nOut from 66 to 16
replace: * 0.01.00              t1._1                   0 f 00240 N 0 cost 0,82,63
    add: * 0.01.00              t1.t1y                  0 f 00262 N 1 cost 0,36,14
```

```
STAT4 range scan: 1..99  est=64
Range scan lowers nOut from 66 to 64
   skip: * 0.01.00              t1.t1x                   0 f 00272 N 2 cost 0,73,63
```

Range запросы

CREATE TABLE tab(id PRIMARY KEY, x, y, z);
CREATE INDEX i1 ON tab(x);
CREATE INDEX i2 ON tab(y);

```
SELECT z FROM tab WHERE
      x BETWEEN 1 AND 100 AND
      y BETWEEN 1 AND 100
```

x IN [0, 1000000] — поиск уменьшается в 10000 раз
y IN [0, 1000] — поиск уменьшается в 10 раз

Skip-Scan

CREATE TABLE people(name PRIMARY KEY, role, age);

CREATE INDEX people_idx1 ON people(role, age);

role IN (‘student’, ‘teacher’)

SELECT name FROM people WHERE age >= 28;

SELECT name FROM people WHERE role = ‘student’ AND age >28

UNION ALL

SELECT name FROM people WHERE role = ‘teacher’ AND age > 28

Skip-Scan

SELECT name FROM people WHERE age >= 28 AND age < 44;

<u>Подсчет стоимости:</u>

nOut = nOut * (min(U - L, 1) / N) — уменьшение кол-ва строк для просмотра

U, L — кол-во sample’ов, которые которые <= 28 и 44

Если разница между граничными условиями мала, то условию удовлетворяет всего 1/64 от всех строк:

nOut = nOut/64

Skip-Scan

Оптимизация отключается, если:

- Среднее число совпадающих элементов в самом левом столбце < 19

- stat1.stat содержит токен “noskipscan”

- Перед условием WHERE стоит унарный плюс:

```
SELECT * FROM t WHERE +x = 3;
```

JOIN

- JOIN использует только объединение циклами
- INNER JOIN — могут меняться местами
- LEFT OUTER JOIN, CROSS JOIN — как в запросе
- Выбор порядка для JOIN —полиномиальный алгоритм

* JOIN == INNER JOIN == COMMA == CROSS JOIN

JOIN

```
CREATE TABLE node(id PRIMARY KEY, name TEXT);
CREATE TABLE edge(source REFERENCES node,
                  dest REFERENCES node);
```

```
CREATE INDEX node_idx ON node(name);
CREATE INDEX edge_idx ON edge(dest, source);
```

```
SELECT * FROM edge AS e, node AS n1, node AS n2 WHERE
    n1.name = ‘alice’ AND
    n2.name = ‘bob’ AND
    e.source = n1.id AND e.dest = n2.id;
```

JOIN


```
foreach n1 where n1.name=‘alice’ do:
    foreach n2 where n2.name=‘bob’ do:
        foreach e where e.source=n1.id and e.source=n2.id do:
                                    ...
```

```
foreach n1 where n1.name=‘alice’ do:
    foreach e where e.source=n1.id do:
        foreach n2 where n2.id=e.dest and n2.name =‘bob’ do:
                                    ...
```

JOIN

случай 1:

Кол-во нод с name = ‘alice’ и name = ‘bob’ == 2

Но много ребер исходящих из них

Опция 1: внутренний цикл выполнится всего 4 раза

Опция 2: гораздо больше итераций

случай 2:

Кол-во нод с name = ‘alice’ и name = ‘bob’ == 3500

Но каждая нода содержит всего 1-2 ребра

Опция 1: внешний и средний цикл по 3500 раз ~ 12 миллионов

Опция 2: внешний цикл 3500 раз, внутренний и средний по 1-2

Авто-индексы

```
CREATE TABLE t1(a PRIMARY KEY, b);
CREATE TABLE t2(c PRIMARY KEY, d);
SELECT * FROM t1, t2 WHERE b = d;
  - - SCAN TABLE t1
  - - SEARCH TABLE t2 USING AUTOMATIC COVERING INDEX (d=?)
```

- Время выполнения без авто-индексов: O(N*N)
- Стоимость создания индекса: X*N*log(N), N — кол-во строк в таблице, X = 7 для обычных таблиц и X = 1.375 для представлений
- Без статистики N ~ 1000000
- Временный индекс — существует только на время запроса

Авто-индексы

```
CREATE TABLE t1(a PRIMARY KEY, b);
CREATE TABLE t2(c PRIMARY KEY, d);

SELECT a, (SELECT d FROM  t2 WHERE c = d) FROM t1;
```

- Если таблица содержит N строк, то ожидается, что подзапрос будет выполнен N раз — O(N^2)
- С авто-индексом — O(N*logN)

Subquery flattening

SELECT a FROM (SELECT x+y AS a FROM t1 WHERE z<100) WHERE a>5

SELECT x+y AS a FROM t1 WHERE z<100 AND a>5

Много условий, которые должны выполняться:

- Не должны присутствовать агрегатные функции
- Не должны присутствовать ограничения (LIMIT)
- Подзапрос должен содержать условие FROM

Subquery

```
SELECT * FROM
(SELECT a AS x, c-d AS y FROM t1)
WHERE x=5 AND y=10;
```

```
SELECT * FROM
(SELECT a AS x, c-d AS y FROM t1 WHERE a=5 AND c-d=10)
WHERE x=5 AND y=10;
```

Subquery Co-routines

- Подзапрос обрабатывается в том же потоке как полноценный запрос

- После вычисления каждой строки подзапроса управление передается основному запросу

- Если нужно использовать результат подзапроса несколько раз, то лучше использовать обычный flattering

Еще Co-routines

```
SELECT f(a) FROM t ORDER BY date DESC LIMIT 5;
```

```
SELECT f(a) FROM (
    SELECT a FROM t ORDER BY date DESC LIMIT 5);
```

MIN/MAX

CREATE INDEX i1 ON tab(x);

SELECT MIN(x) FROM tab;

SELECT MIN(x + 1) FROM tab;

SELECT MIN(x) + 1 FROM tab;

Индексы для выражений

```
CREATE TABLE t2(x PRIMARY KEY, y, z);

CREATE INDEX t2xy ON t2(x+y);
```

```
SELECT * FROM t2 WHERE y+x = 22;
```

```
SELECT * FROM t2 WHERE x+y = 22;
```

Чек-лист

- НЕ ПАНИКОВАТЬ!
- Создавать индексы
- Создавать хорошие индексы
- ANALYZE
- EXPLAIN QUERY PLAN
- Ручное управление